# Исходные коды

программного комплекса:

«Petri Nets»

Выполнили: студент группы ИВ-73

НТУУ «КПИ» ФИВТ
Ашаев Юрий
Грубый Павел
Новотарский Кирилл
Пустовит Михаил

***Киев-2009***

# Оглавление

# \data

## \data\Bridge.java

```
package ua.com.handcrafted.petri.data;

import java.util.ArrayList;
import java.util.HashMap;


/**
 * @author Pustovit Michael
 * Класс предстваляющий переход в модели данных.
 */

public class Bridge extends Node {
    /**
     * @author Pustovit Michael
     * Класс "связь" - представляет связь перехода и состояния в
     * модели данных.
     */
    class Link {
        private State end;

        public Link(State end) {
            this.end = end;
        }

        public State getEnd() {
            return end;
        }

        public void setEnd(State end) {
            this.end = end;
        }

    }


    /** List of input links. */
    private ArrayList<Link> inLink;

    /** ArrayList of output links. */
    private ArrayList<Link> outLink;


    //Конструкторы
    public Bridge() {
        inLink = new ArrayList<Link>();
        outLink = new ArrayList<Link>();
    }

    public Bridge(ArrayList<Link> inLink, ArrayList<Link> outLink) {
        this.inLink = inLink;
        this.outLink = outLink;
    }

    /**
     * @param state Состояние из которого выходит связь.
     * Добавление входной связи (стрелка направлена к переходу).
     */
    public void addInLink(State state) {
        inLink.add(new Link(state));
    }

    /**
     * @param state Состояние в которое входит связь.
     * Добавление выходной связи (стрекла направлена от перехода).
     */
    public void addOutLink(State state) {
```

```
            outLink.add(new Link(state));
        }

        /**
         * @return Разрешен ли данный переход в текущий момент времени.
         */
        public boolean isAllow() {
            boolean flag = true;
            HashMap<State, Integer> temp = new HashMap<State, Integer>();
            for (int i = 0; (i < inLink.size()) && flag; i++) {
                if (temp.containsKey(inLink.get(i).getEnd())) {
                    int t = temp.get(inLink.get(i).getEnd()).intValue();
                    if (t + 1 > inLink.get(i).getEnd().getChips()) {
                        flag = false;
                    } else {
                        temp.put(inLink.get(i).getEnd(), t + 1);
                    }
                } else {
                    if (inLink.get(i).getEnd().getChips() != 0) {
                        temp.put(inLink.get(i).getEnd(), 1);
                    } else {flag = false;}
                }
            }

            return flag;
        }

        /**
         * @return Список входных связей.
         * Получение списка входных связей.
         */
        public ArrayList<Link> getInLink() {
            return inLink;
        }

        /**
         * @return Список выходных связей.
         * Получение списка выходных связей.
         */
        public ArrayList<Link> getOutLink() {
            return outLink;
        }
    }
```

## \data\DataModel.java

```
package ua.com.handcrafted.petri.data;

import java.util.ArrayList;

import ua.com.handcrafted.petri.matrix.Matrix;
import ua.com.handcrafted.petri.matrix.Vector;
import ua.com.handcrafted.petri.processing.Marking;

/**
 * @author Pustovit Michael
 * Data model of Petri-Nets. It consists of lists of Bridges and States.
 */
public class  DataModel {
    /** List of nodes. */
    private ArrayList<State> stateList;

    /** ArrayList of bridges */
    private ArrayList<Bridge> bridgeList;
    private Matrix InMatrix, OutMatrix, ResultMatrix;

    public DataModel(ArrayList<State> nodeList, ArrayList<Bridge> bridgeList) {
        this.stateList = nodeList;
        this.bridgeList = bridgeList;
    }
```

```
public DataModel() {
    this.stateList = new ArrayList<State>();
    this.bridgeList = new ArrayList<Bridge>();
}

/**
 * Построение матриц входов-переходов, выходов-переходов и результирующей
 * матрицы. Если в модели нет состояний то будет ошибка.
 */
public void initializeMatrix() {
    InMatrix = processInMatrix();
    OutMatrix = processOutMatrix();
    ResultMatrix = processResultMatrix();
}

/**
 * Добавлене состояния в модель.
 * @param state Добавляемое состояние.
 */
public void addState(State state) {
    stateList.add(state);
}

/**
 * Добавление перехода в модель.
 * @param bridge Добавляемый переход.
 */
public void addBridge(Bridge bridge) {
    bridgeList.add(bridge);
}

/**
 * Снятие текущей маркировки с модели.
 * @return Текущая маркировка.
 */
public ArrayList<Integer> getMarking() {
    ArrayList<Integer> res = new ArrayList<Integer>();
    for (int i = 0; i < stateList.size(); i++) {
        res.add(stateList.get(i).getChips());
    }
    return res;
}

/**
 * Установка маркировки на модель.
 * @param mark Устанавливаемая маркировка.
 */
public void setMarking(Marking mark) {
    for (int i = 0; i < stateList.size(); i++) {
        stateList.get(i).setChips(mark.getMarking().get(i));
    }
}

/**
 * Список разрешенных мгновенных переходов.
 * @return Разрешенные мгновенные переходы.
 */
public ArrayList<Bridge> getAllowInstantBridges() {
    ArrayList<Bridge> res = new ArrayList<Bridge>();
    for (int i = 0; i < bridgeList.size(); i++) {
        if ((bridgeList.get(i).isAllow())
                && (bridgeList.get(i).getClass() != TimeBridge.class))
            res.add(bridgeList.get(i));
    }
    return res;
}

/**
 * Список разрешенных временных переходов.
```

```
     * @return Разрешенные временные переходы.
     */
    public ArrayList<Bridge> getAllowTimeBridges() {
        ArrayList<Bridge> res = new ArrayList<Bridge>();
        for (int i = 0; i < bridgeList.size(); i++) {
            if ((bridgeList.get(i).isAllow())
                    && (bridgeList.get(i).getClass() == TimeBridge.class))
                res.add(bridgeList.get(i));
        }
        return res;
    }

    /**
     * @return Разрешенные на данный момент мгновенные переходы.
     */
    public Vector getAllowInstantBridgesVector() {
        ArrayList<Bridge> allowArray = getAllowInstantBridges();
        int [] rez = new int [bridgeList.size()];
        for (int i = 0; i < allowArray.size(); i++) {
            rez[bridgeList.indexOf(allowArray.get(i))] = 1;
        }
        return new Vector(rez);
    }

    /**
     * @return Входная матрица
     * Вычисление (по модели данных) входной матрицы.
     */
    private Matrix processInMatrix() {
        int [][] mas = new int [bridgeList.size()][stateList.size()];
        for (int i = 0; i < bridgeList.size(); i++) {
            for (int j = 0; j < bridgeList.get(i).getInLink().size(); j++) {
                int help =
stateList.indexOf(bridgeList.get(i).getInLink().get(j).getEnd());
                mas[i][help]++;
            }
        }
        return new Matrix(mas);
    }

    /**
     * @return Выходная матрица
     * Вычисление (по модели данных) выходной матрицы.
     */
    private Matrix processOutMatrix() {
        int [][] mas = new int [bridgeList.size()][stateList.size()];
        for (int i = 0; i < bridgeList.size(); i++) {
            for (int j = 0; j < bridgeList.get(i).getOutLink().size(); j++) {
                int help =
stateList.indexOf(bridgeList.get(i).getOutLink().get(j).getEnd());
                mas[i][help]++;
            }
        }
        return new Matrix(mas);
    }


    /**
     * @param bridge Переход номер которого мы ищем.
     * @return Номер перехода.
     * Поиск номера перехода в списке переходов модели данных.
     */
    public int getBridgeIndex(Bridge bridge) {
        return bridgeList.indexOf(bridge);
    }

    /**
     * @param index Индекс по которому мы хотим получить переход.
     * @return Переход по заданному индексу.
     * Возвращает переход который хранится по индексу index в списке
```

```
     * переходов модели данных.
     */
    public Bridge getBridgeByIndex(int index){
       return bridgeList.get(index);
    }

    /**
     * @param index Индекс по которому мы хотим получить состояние.
     * @return Состояние по заданному индексу.
     * Возвращает состояние которое хранится по индексу index в списке
     * состояний модели данных.
     */
    public State getStateByIndex(int index){
       return stateList.get(index);
    }

    /**
     * @return Количество переходов в модели.
     */
    public int getBridgeCount(){
       return bridgeList.size();
    }

    /**
     * @return Количество состояний в программе.
     */
    public int getStateCount(){
       return stateList.size();
    }


    /**
     * @return Вычисление результирующей матрицы.
     */
    private Matrix processResultMatrix() {
        return OutMatrix.sub(InMatrix);
    }

    /**
     * @return Входная матрица.
     */
    public Matrix getInMatrix() {
        return InMatrix;
    }

    /**
     * @return Выходная матрица.
     */
    public Matrix getOutMatrix() {
        return OutMatrix;
    }

    /**
     * @return Результирующая матрица.
     */
    public Matrix getResultMatrix() {
        return ResultMatrix;
    }

    /**
     * @return Пуст ли список состояний.
     * Проверка на пустоту списка переходов
     * (используется в "защите от дурака": модель пуста - нельзя переходить
     * в режим моделирования).
     */
    public boolean isStateListEmpty() {
        return stateList.isEmpty();
    }
}
```

## \data\Node.java

```
package ua.com.handcrafted.petri.data;


/**
 * @author Pustovitm Michael
 * Абстрактный класс от которого наследуются все остальные сущности модели
 * данных.
 */
public abstract class Node {

    /**
     * @uml.property  name="active"
     */
    private boolean active = false;

    /**
     * Getter
 of the property <tt>active</tt>
     * @return  Returns the active.
     * @uml.property  name="active"
     */
    public boolean isActive() {
        return active;
    }

    /**
     * Setter of the property <tt>active</tt>
     * @param active  The active to set.
     * @uml.property  name="active"
     */
    public void setActive(boolean active) {
        this.active = active;
    }

    /**
     * @uml.property  name="id"
     */
    private String id;

    /**
     * Getter of the property <tt>id</tt>
     * @return  Returns the id.
     * @uml.property  name="id"
     */
    public String getId() {
        return id;
    }

    /**
     * Setter of the property <tt>id</tt>
     * @param id  The id to set.
     * @uml.property  name="id"
     */
    public void setId(String id) {
        this.id = id;
    }
}
```

## \data\State.java

```
package ua.com.handcrafted.petri.data;


/**
 * @author Pustovit Michael
 * Класс представляющий состояние в модели данных.
 *
 */
```

```
public class State extends Node {

    /**
     * @param chips Количество фишек в данном состоянии.
     */
    public State(int chips) {
        this.chips = chips;
        maxChips = -1;
    }

    public State() {
        this.chips = 0;
        maxChips = -1;
    }

    /**
     * @param chips Количество фишек в данном состоянии.
     * @param maxChips Максимальное количество фишек в данном состоянии.
     */
    public State(int chips, int maxChips) {
        this.chips = chips;
        this.maxChips = maxChips;
    }

    /**
     * @uml.property  name="chips"
     * Count of chips in this state.
     */
    private int chips = 0;

    /**
     * Maximum count of chips in this State.
     * -1 = infinity
     */
    private int maxChips = -1;

    /**
     * Getter of the property <tt>chips</tt>
     * @return  Returns the chips.
     * @uml.property  name="chips"
     */
    public int getChips() {
        return chips;
    }

    /**
     * Setter of the property <tt>chips</tt>
     * @param chips  The chips to set.
     * @uml.property  name="chips"
     */
    public void setChips(int chips) {
        this.chips = chips;
    }

    /**
     * @return Максимальное количество фишек в данном состоянии.
     */
    public int getMaxChips() {
        return maxChips;
    }

    /**
     * @param max Максимальное количество фишек в данной вершине.
     * Установка максимального количества фишек.
     */
    public void setMaxChips(int max) {
        maxChips = max;
    }

    /**
```

```
     * Уменьшить количество фишек.
     */
    public void decCheaps() {
        chips--;
    }

    /**
     * Увеличить количество фишек.
     */
    public void incCheaps() {
        chips++;
    }
}
```

## \data\TimeBridge.java

```
package ua.com.handcrafted.petri.data;

import ua.com.handcrafted.petri.generator.Generator;


/**
 * @author Pustovit Michael
 * Временной переход
 */
public class TimeBridge extends Bridge {
    private double lambda;
    private double r;
    private String g;
    private double worktime = -1;
    private Generator gen;

    /**
     * @param lambda Параметр генератора
     * @param r Вероятность с которой выбирается данный переход в случае
     * розыгрыша.
     * @param g Параметр генератора.
     */
    public TimeBridge(double lambda, double r, String g) {
        super();
        this.lambda = lambda;
        this.g = g;
        this.r = r;
        gen = new Generator(lambda, g);
    }

    public TimeBridge() {
        super();
    }

    public void setLambda(double lambda) {
        this.lambda = lambda;
    }
    public double getLambda() {
        return lambda;
    }

    public void setR(double r) {
        this.r = r;
    }
    public double getR() {
        return r;
    }

    public void setG(String g) {
        this.g = g;
    }
    public String getG() {
        return g;
    }
```

```
    /**
     * @param worktime Время за которое сработает переход.
     * Установка времени работы перехода.
     */
    public void setWorktime(double worktime) {
        this.worktime = worktime;
    }

    /**
     * @return Получить время работы перехода.
     */
    public double getWorktime() {
        return worktime;
    }

    /**
     * Сгененрировать новое время работы перехода.
     */
    public void setNewWorktime() {
        if (worktime == -1) {
            worktime = gen.getNext();
        }
    }

    /**
     * Выставление времени работы в -1, что означает что данный переход
     * не находится в данный момент в очерди на срабатывание и не имеет
     * времени срабатывания.
     */
    public void unsetWorktime() {
        worktime = -1;
    }
}
```

# \generator

## \generator\Generator.java

```
package ua.com.handcrafted.petri.generator;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Random;

public class Generator{
       private int nVal;
       private double[] model;
       private int[] intervals;
       /**
        * Возвратить кол-во значений
        * @return
        */
       public int getnVal() {
              return nVal;
       }
       private int nInt;
       /**
        * Возвратить кол-во интервалов
        * @return кол-во интервалов
        */
       public int getnInt() {
              return nInt;
       }
       /**
        * Установить кол-во интервалов
        * @param nInt новое кол-во интервалов
```

```
	 */
	public void setnInt(int nInt) {
		this.nInt = nInt;
		this.intervals=new int[nInt];
	}
	/**
	 * Установить количество значений
	 * @param nVal количество значений
	 */
	public void setnVal(int nVal) {
		this.nVal = nVal;
		this.model=new double[nVal];
	}

	/**
	 * Заполнить массив количеств чисел в интервалах
	 */
	public void genInt() {
		double max=maxVal();
		intervals = new int[nInt];
		double intLen = max / nInt;
		for (int i=0;i<nVal;i++){
			int numInt = new
BigDecimal(model[i]/intLen).setScale(0,RoundingMode.DOWN).intValue();
			if ((numInt < nInt) && (numInt >= 0)) {
				intervals[numInt]=intervals[numInt]+1;
			}
		}
	}
	/**
	 * Получить значение чисел в заданном интервале
	 * @param i
	 * @return кол-во чисел
	 */
	public int getInterval(int i) {
		return intervals[i];
	}

	/**
	 * Конструктор
	 * @param nVal - Кол-во значений в выборке
	 * @param nInt - Кол-во интервалов
	 */
	Generator(int nVal, int nInt){
		this.nInt=nInt;
		this.nVal=nVal;
		this.model=new double[nVal];
		this.intervals=new int[nInt];
	}
	/**
	 * Сгенерировать выборку в 10000 чисел с заданными
	 * параметрами лямбда и ковариации. Лямбда и ковариация - дробные числа.
	 * Лямбда задается числом, ковариация - строкой. Специальное значение "1/3"
	 * предусмотрено для равномерного потока.
	 * @param lambda
	 * @param g
	 */
	public Generator(double lambda, String g){
		this.nInt=20;
		this.nVal=10000;
		this.model=new double[nVal];
		this.intervals=new int[nInt];
		double a = Math.exp(4*Math.log(93));
		double c = Math.exp(4*Math.log(91));
		double d = Math.exp(4*Math.log(19));
		double W0 = 0.5;
		gen(a, c, d, W0);
		genInRandom();
		if (g.equals("1/3")){
			genUniformStream(lambda);
```

```
		} else if (g.equals("1")){
			genExp(lambda);
		} else {
			double var = Double.valueOf(g);
			if (var < 1){
				genErlanga(lambda, (int) (1/var));
			} else {
				genHypExp(lambda, var);
			}
		}
	}



	/**
	 * Перегенерировать последовательность
	 * внутренним генератором случайных чисел
	 */
	public void genInRandom(){
		Random rd = new Random();
		for (int i=0;i < nVal;i++){
			model[i]=rd.nextDouble();
		}
		genInt();
	}
	/**
	 * Перегенерировать последовательность
	 * @param A - параметр генератора
	 * @param C - параметр генератора
	 * @param D - параметр генератора
	 * @param W0 - параметр генератора
	 */
	public void gen(double A,double C,double D, double W0){
		double Wi = W0;
		int i;
		for (i=0;i < nVal;i++){
			model[i]=Wi;
			Wi=(A*Wi+C)/D%1;
		}
		genInt();
	}
	/**
	 * Перегенерировать главную последовательность с
	 * помощью аддитивного генератора
	 * @param k
	 */
	public void genAdd(int k){
		double sum;
		for (int i=k; i<nVal;i++){
			sum = 0;
			for (int j=i-k;j<i;j++){
				sum = sum + model[j];
			}
			model[i] = sum % 1;
		}
		genInt();
	}
	/**
	 * Перегенерировать главную последовательность с
	 * помощью экспоненциального генератора
	 * @param l - параметр генератора
	 *
	 */
	public void genExp(double l){
		for (int i=0; i<nVal;i++){
			model[i] = (-Math.log(1-model[i])/l);
		}
		genInt();
	}
	/**
```

```
 * Перегенерировать главную последовательность с
 * помощью гиперэкспоненциального генератора
 * @param l - параметр генератора
 * @param g - параметр генератора
 */
public void genHypExp(double l,double g){
	double phi = 0.5 - Math.sqrt(0.25-(double)1/(2*g+2));
	double a = 0;
	for (int i=1; i<nVal-1;i++){
		if (phi > model[i]){
			a = 2*phi*l;
		} else{
			a = 2*(1-phi)*l;
		}
		model[i] = (-Math.log(model[i+1])/a);
	}
	genInt();
}

/**
 * Перегенерировать главную последовательность с
 * помощью генератора Эрланга
 * @param lambda - параметр генератора
 * @param k - параметр генератора
 */
public void genErlanga(double lyambda,int k){
	double[] buf = new double[model.length/k];
    for (int i = 0; i<model.length/k ; i++) {
	   double s=0;
	   for (int j = 0; j < k; j++) {
			s=s+Math.log(model[i*k + j]);
		}
		buf[i]=s*(-1/lyambda/k);
	}
    model = buf;
    this.nVal = model.length/k;
	   genInt();
}
/**
 * Генератор Эрланга с сайта
 * http://algolist.manual.ru/maths/matstat/erlang/index.php
 * @param lyambda
 * @param a
 */
public void genErlanga2(double lyambda,int a){
	double b = 1/lyambda;
	double newVal = 0;
	for (int i = 0; i<nVal ;i++) {
	double s=1;
	for (int j = 0; j < a; j++) {
			s=s*Math.random();
		}
	newVal = -b*Math.log(s);
	    model[i]= newVal;
	}
	genInt();
}

/**
 * Равномерное распределение
 * @param lamda
 */
public void genUniformStream(double lamda){
	for (int i=0; i<nVal;i=i+1){
		model[i] = 2*model[i]/lamda;
	}
	genInt();
}
/**
 * Проверка
```

```
на взаимную простоту
	 * @param a
	 * @param b
	 * @param c
	 * @return - есть ли они простые
	 */
	public boolean isSimply(int a,int b, int c){
		int max = a;
		if (b>max) {max=b;};
		if (c>max) {max=c;};
		for (int i=2; i<=max; i++){
			int k = 0;
			if (a % i == 0) {k++;}
			if (b % i == 0) {k++;}
			if (c % i == 0) {k++;}
			if (k>1) {
				return false;
			}
		}
		return true;
	}
	/**
	 * Найти интервал, содержащий наибольшее
	 * кол-во чисел
	 * @return число
	 */
	public double maxInter(){
		double Buf;
		Buf = intervals[0];
		for (int i=1;i<nInt;i++){
			if (intervals[i]>Buf){
				Buf=intervals[i];
			}
		}
		return Buf;
	}
	/**
	 * Найти максимальное число
	 * @return число
	 */
	public double maxVal(){
		double Buf;
		Buf = model[0];
		for (int i=1;i<nVal;i++){
			if (model[i]>Buf){
				Buf=model[i];
			}
		}
		return Buf;
	}
	/**
	 * Найти интервал, содержащий наименшее
	 * кол-во чисел
	 * @return число
	 */
	public double minInter(){
		double Buf;
		Buf = intervals[0];
		for (int i=1;i<nInt;i++){
			if (intervals[i]<Buf){
				Buf=intervals[i];
			}
		}
		return Buf;
	}
	/**
	 * Найти число с максимальным
	 * отклонением от среднего
	 * @return число
	 */
```

```
	public double maxP(){
		double Buf,avgE,maxP;
		avgE = avgVal();
		maxP = Math.abs(intervals[0]-avgE);
		for (int i=1;i<nInt;i++){
			Buf = Math.abs(intervals[i]-avgE);
			if (maxP<Buf){
				maxP=Buf;
			}
		}
		maxP = maxP/avgE;
		return maxP;
	}
	/**
	 * Среднее значение
	 * @return среднее
	 */
	public double avgVal(){
		double Buf;
		Buf = 0;
		for (int i=0;i<nInt;i++){
			Buf=Buf+intervals[i];
		}
		return Buf/nInt;
	}
	/**
	 *
	 * @return матожидание
	 */
	public double expectationVal(){
		double mR=0;
		for (int i=0; i<nVal; i++){
			mR=mR+model[i];
		}
		mR=mR/nVal;
		return mR;
	}
	/**
	 * Коефициент вариации потока
	 * @return
	 */
	public double variation(){
		double m = expectationVal();
		return dispersion()/(m*m);
	}
	/**
	 *
	 * @return дисперсия
	 */
	public double dispersion(){
		double mR=expectationVal();
		double disp=0;
		for (int i=0; i<nVal; i++){
			disp=disp+(model[i]-mR)*(model[i]-mR);
		}
		disp=disp/nVal;
		return disp;
	}
	/**
	 * Возвращает случайное число с сгенерированой выборки.
	 * @return
	 */
	public double getNext(){
		int num = (int)(Math.random()*nVal);
		return model[num];
	}
}
```

### \generator\GenTest.java

```
package ua.com.handcrafted.petri.generator;

/**
 * Пример работы с генератором
 * @author Ашаев Юрий
 *
 */
public class GenTest{
	public static void main(String[] args){
		Generator gen = new Generator(1,"1/3");
		System.out.println("variation="+gen.variation());
		System.out.println("number="+gen.getNext());
	}
}
```

## \matrix

### \matrix\IntegerMatrix.java

```
package ua.com.handcrafted.petri.matrix;

/**
 * @author Someone
 * Матрица из Целых
 *
 */
public class IntegerMatrix {
	private Matrix matrix;
	private Integer[][] objectMatrix;

	public IntegerMatrix(Matrix matrix) {
		super();
		this.matrix = matrix;
		int[][] m = matrix.getMatrix();
		for (int i = 0; i<matrix.getRowDimension(); i++) {
			for (int j = 0; i<matrix.getColumnDimension(); j++) {

			}

		}
	}

}
```

### \matrix\Matrix.java

```
package ua.com.handcrafted.petri.matrix;

/**
 * @author Pustovit Michael
 * Матрица созданная для использования при создании матриц входов-переходов,
 * выходов-переходов, а так же результирующей матрицы.
 */
public class Matrix {
    private int [][] matrix;

    public Matrix(int [][] mas) {
        this.matrix = mas;
    }

    /**
     * @return Количество строк матрицы.
     */
    public int getRowDimension() {
        return matrix.length;
```

```
    }

    /**
     * @return Количество столбцов матрицы.
     */
    public int getColumnDimension() {
        if (matrix.length > 0) return matrix[0].length;
        else return 0;
    }

    public int [][] getMatrix() {
        return matrix;
    }

    /**
     * Вичитание от данной матрицы переданной.
     * @param mas матрица которая будет отниматься.
     * @return результат вычитания матриц.
     */
    public Matrix sub(Matrix mas) {
       int [][] rez = new int [getRowDimension()][getColumnDimension()];
       for (int i = 0; i < getRowDimension(); i++) {
           for (int j = 0; j < getColumnDimension(); j++) {
              rez[i][j] = matrix[i][j] - mas.getMatrix()[i][j];
           }
       }
       return new Matrix(rez);
    }

    public String toString() {
        String str = "";
        for (int i = 0; i < getRowDimension(); i++) {
            for (int j = 0; j < getColumnDimension(); j++) {
                str = str + matrix[i][j] + "  ";
            }
            str = str + "\n";
        }
        return str;
    }
}
```

## \matrix\Vector.java

```
package ua.com.handcrafted.petri.matrix;

/**
 * @author Pustovit Michael
 *  Вектор созданный для организации векторов разрешенных переходов
 *  (на них умножается результирующая матрица).
 */
public class Vector {
    private int [] vector;

    public Vector(int [] mas) {
        this.vector = mas;
    }

    /**
     * Create vector which consists of 0 except pos position, which equals 1.
     * @param pos
     */
    public Vector(int pos, int length) {
        vector = new int [length];
        vector[pos] = 1;
    }

    public int getDimension() {
        return vector.length;
    }
```

```
public int [] getVector() {
    return vector;
}

/**
 * Умножение данного вектора на переданный.
 * @param mas Вектор на который умножаем.
 * @return Результат умножения.
 */
public Vector multiple(Matrix mas) {
    int [] vect = new int [mas.getColumnDimension()];
    for (int i = 0; i < mas.getColumnDimension(); i++) {
        for (int j = 0; j < vector.length; j++) {
            vect[i] = vect[i] + vector[j] * mas.getMatrix()[j][i];
        }
    }

    return new Vector(vect);
}

/**
 * Сложение векторов.
 * @param vect Вектор который прибавляем.
 * @return Результат суммирования.
 */
public Vector add(Vector vect) {
    int [] rez = new int [vector.length];
    for (int i = 0; i < vector.length; i++) {
        rez[i] = vector[i] + vect.getVector()[i];
    }
    return new Vector(rez);
}

/**
 * Проверка не содержит ли данный вектор одни 0.
 * @return Если все - 0, то true, иначе - false.
 */
public boolean isEmpty() {
    boolean flag = true;
    for (int i = 0; (i < vector.length) && (flag); i++) {
        if (vector[i] != 0) flag = false;
    }
    return flag;
}

/**
 * Заполнение вектора нулями.
 */
public void clear() {
    for (int i = 0; i < vector.length; i++) {
        vector[i] = 0;
    }
}

@Override
public String toString() {
    String str = "";
    for (int i = 0; i < getDimension(); i++) {
        str = str + vector[i] + "  ";
    }
    return str;
}

@Override
public boolean equals(Object obj) {
    Vector vect = (Vector) obj;
    boolean flag = true;
    for (int i = 0; i < vector.length; i++) {
        if (vect.getVector()[i] != vector[i]) flag = false;
    }
```

```
        return flag;
    }


}
```

# \modelling

## \modelling\Experiment.java

```
package ua.com.handcrafted.petri.modelling;

import java.util.ArrayList;

import ua.com.handcrafted.petri.data.Bridge;
import ua.com.handcrafted.petri.data.DataModel;
import ua.com.handcrafted.petri.data.TimeBridge;
import ua.com.handcrafted.petri.matrix.Vector;
import ua.com.handcrafted.petri.processing.BuildTree;
import ua.com.handcrafted.petri.processing.Marking;

/**
 * @author Pustovit Michael
 * Класс для моделирования процесса на сети Петри.
 */
public class Experiment {

    /**
     * @author Pustovit Michael
     * Класс "пара". Содержит пару состояний: первое - из которого ушла фишка,
     * второе - в которое фишка пришла. После процесса моделирования, по массиву
     * таких пар будет строится матрицы вероятности перехода.
     */
    private class Couple {
        public Marking first;
        public Marking second;

        public Couple(Marking f, Marking s) {
            first = f;
            second = s;
        }
    }

    /**
     * Используемая модель данных.
     */
    private DataModel model;
    /**
     * Количество шагов моделирования.
     */
    private Integer exCount;
    /**
     * Максимальное время моделирования.
     */
    private Double exTime;
    /**
     * Таблица переходов (сколько раз перешли из i-ого состояния в j-ое)
     */
    private int[][] crossingTable;
    /**
     * Таблица вероятности переходов.
     */
    private double[][] probabilityTable;
    private double dt = 1; // Единица времени
    /**
     * Маркировка с которой начинается моделирование.
     */
```

```
    private Marking fM; // начальная маркировка
    /**
     * Текущее время моделирования.
     */
    private double allTime = 0;

    /**
     * @param model
     *            Модель данных на которой будет производится моделирование.
     * @param exCount
     *            Количество шагов моделирования.
     * @param tree
     *            Начальная вершина дерева достижимости.
     */
    public Experiment(DataModel model, int exCount, double exTime) {
        this.model = model;
        this.exCount = exCount;
        this.exTime = exTime;
        this.fM = new Marking(model.getMarking());
    }

    /**
     * @return Массив "строк" статистической таблицы. Каждая "строка" это
     * класс StatLine который содержит все статистические данные по одной
     * маркировке.
     * Старт процесса моделирования.
     */

    public ArrayList<StatLine> start() {
        // выставление начальной маркировки
        Marking cM = fM.clone();
        // статистическая
таблица
        ArrayList<StatLine> stat = new ArrayList<StatLine>();
        //очередь разрешенных переходов переходов
        ArrayList<TimeBridge> queue = new ArrayList<TimeBridge>();
        //массив пар, по которому в последствии будет строится матрица переходов
        ArrayList<Couple> pig = new ArrayList<Couple>();

        for (int i = 0; (i < exCount) &&
            ((allTime < exTime) || (exTime < 0)); i++) {
            StatLine tempStat = new StatLine(cM);
            if (stat.contains(tempStat)) {
                tempStat = stat.get(stat.indexOf(tempStat));
            } else {
                stat.add(tempStat);
            }
            tempStat.incCountAppear();
            tempStat.incLackTime(allTime - tempStat.getLastTime());

            addNewBridges(queue);
            if (queue.size() == 0) {
                System.out.println("Тупиковая маркировка");
                break;
            }
            clearColision(queue);
            for (int j = 0; j < queue.size(); j++) {
                // double rand = Math.random();
                queue.get(j).setNewWorktime();
            }
            TimeBridge curBri = getMinTime(queue);
            queue.remove(curBri);
            allTime += curBri.getWorktime();
            tempStat.incSumTime(curBri.getWorktime());
            tempStat.setLastTime(allTime);

            for (int j = 0; j < queue.size(); j++) {
                queue.get(j).setWorktime(
                        queue.get(j).getWorktime() - curBri.getWorktime());
            }
```

```
            curBri.unsetWorktime();

            BuildTree builder = new BuildTree(cM, model);
            Vector vect = new Vector(model.getBridgeIndex(curBri), model
                    .getBridgeCount());
            ArrayList<Marking> tempMark = builder.getInstantSon(cM, vect,
                    new ArrayList<Vector>());

            if (tempMark.size() == 1) {
                pig.add(new Couple(cM, tempMark.get(0)));
                cM = tempMark.get(0);
            } else {
                // error
            }
        }

        ArrayList<Marking> temp = new ArrayList<Marking>();
        for (int i = 0; i < stat.size(); i++) {
            temp.add(stat.get(i).getMark());
        }

        crossingTable = new int[stat.size()][stat.size()];
        for (int i = 0; i < pig.size(); i++) {
            if ((temp.indexOf(pig.get(i).first) >= 0)
&&temp.indexOf(pig.get(i).second) >= 0)

{crossingTable[temp.indexOf(pig.get(i).first)][temp.indexOf(pig.get(i).second)]++;}
        }

        double [] dTarr = new double [stat.size()];
        for (int i = 0; i < stat.size(); i++) {
            int sum = 0;
            for (int j = 0; j < stat.size(); j++) {
                sum += crossingTable[i][j];
            }
            dTarr[i] = stat.get(i).getSumTime() /(double) sum;
        }

        int min = 0;
        for (int i = 0;  i < dTarr.length; i++) {
            if (dTarr[i] < dTarr[min]) {min = i;}
        }

        dt = dTarr[min];

        probabilityTable = new double[stat.size()][stat.size()];
        for (int i = 0; i < crossingTable.length; i++) {
            for (int j = 0; j < crossingTable[0].length; j++) {
                if (i != j) {probabilityTable[i][j] = crossingTable[i][j]
                            / stat.get(i).getSumTime() * dt;}
            }
        }
        double tempT;
        for (int i = 0; i < crossingTable.length; i++) {
            tempT = 0;
            for (int j = 0; j < crossingTable[0].length; j++) {
                tempT += probabilityTable[i][j];
            }
            probabilityTable[i][i] = 1 - tempT;
        }

        return stat;
    }

    /**
     * @param queue Очередь разрешенных временных переходов.
     * Разрешение коллизий - если условием срабатывания двух переходов
     * является наличие фишки в одном состоянии, то производится розыгрыш
     * в результате которого решается какой из переходов сработает.
     */
```

```
    private void clearColision(ArrayList<TimeBridge> queue) {
        ArrayList<TimeBridge> temp = new ArrayList<TimeBridge>();
        for (int i = 0; i < model.getInMatrix().getColumnDimension(); i++) {
            temp.clear();
            for (int j = 0; j < model.getInMatrix().getRowDimension(); j++) {
                if ((model.getInMatrix().getMatrix()[j][i] == 1)
                        && (model.getBridgeByIndex(j).getClass() == TimeBridge.class))
{
                    temp.add((TimeBridge) model.getBridgeByIndex(j));
                }
            }
            boolean flag = true;
            if (temp.size() > 1) {
                for (int j = 0; j < temp.size(); j++) {
                    if (!queue.contains(temp.get(j))) {
                        flag = false;
                    }
                }
                if (flag) {
                    double sum = 0;
                    for (int j = 0; j < temp.size(); j++) {
                        sum += temp.get(j).getR();
                    }
                    for (int j = 0; j < temp.size(); j++) {
                        temp.get(j).setR(temp.get(j).getR() / sum);
                    }

                    double rand = Math.random();
                    int rez = whichBridge(temp, rand);

                    for (int j = 0; j < temp.size(); j++) {
                        if (j != rez) {
                            temp.get(j).unsetWorktime();
                            queue.remove(temp.get(j));
                        }
                    }
                }
            }
        }
    }

    /**
     * @param temp Переходы участвующие в розыгрыше.
     * @param rand Число которое выдал генератор случайных чисел.
     * @return Какой из переходов выиграл розыгрыш.
     * Определяет какой из переходов выиграл розыгрыш.
     */
    private int whichBridge(ArrayList<TimeBridge> temp, double rand) {
        int i = 0;
        double sum = temp.get(0).getR();
        for (i = 1; rand > sum; i++) {
            sum += temp.get(i).getR();
        }
        return i - 1;
    }

    /**
     * @param queue Очередь разрешенных переходов.
     * Добавляет в очередь разрешенных переходов появившиеся разрешенные
     * переходы (если такие есть).
     */
    private void addNewBridges(ArrayList<TimeBridge> queue) {
        ArrayList<Bridge> temp = model.getAllowTimeBridges();
        for (int i = 0; i < temp.size(); i++) {
            if (!queue.contains((TimeBridge) temp.get(i))) {
                queue.add((TimeBridge) temp.get(i));
                // queue.get(i).setWorktime(Math.random());
            }
        }
    }
```

```
    /**
     * @param queue Очередь разрешенных временных переходов.
     * @return Временной переход с наименьшим временем срабатывания.
     */
    private TimeBridge getMinTime(ArrayList<TimeBridge> queue) {
        TimeBridge min = queue.get(0);
        for (int i = 1; i < queue.size(); i++) {
            if (min.getWorktime() > queue.get(i).getWorktime()) {
                min = queue.get(i);
            }
        }
        return min;
    }

    public int[][] getCrossingTable() {
        return crossingTable;
    }

    public void setExCount(Integer exCount) {
        this.exCount = exCount;
    }

    public Integer getExCount() {
        return this.exCount;
    }

    public void setDt(double dt) {
        this.dt = dt;
    }

    public double getDt() {
        return this.dt;
    }

    /**
     * @return Общее время моделирования.
     */
    public double getAllTime() {
        return allTime;
    }

    public double getProbability(StatLine analyze) {
        double result = analyze.getSumTime() / getAllTime();
        return result;
    }

    public double getFrequency(StatLine analyze) {
        double result = analyze.getSumTime() / getDt();
        return result;
    }

    public double[][] getProbabilityTable() {
        return probabilityTable;
    }
}
```

## \modelling\StatLine.java

```
package ua.com.handcrafted.petri.modelling;

import ua.com.handcrafted.petri.processing.Marking;

/**
 * @author Pustovit Michael
 * Строка таблицы статистики результатов моделирования.
 */
public class StatLine {
      private Marking mark;
```

```
    /**
     * Количество появлений в данной маркировке.
     */
    private int countAppear;

    /**
     * Суммарное время пребывания в данной маркировке.
     */
    private double sumTime;

    /**
     * Суммарное время возвращения.
     */
    private double lackTime;

    /**
     * Момент времени в который мы последний, на текущий момент,
     * раз ушли из маркировки.
     */
    private double lastTime;

    public StatLine(Marking mark) {
           this.mark = mark;
           countAppear = 0;
           sumTime = 0;
           lackTime = 0;
           lastTime = 0;
    }

    public int getCountAppear() {
           return countAppear;
    }

    /**
     * @return the mark
     */
    public Marking getMark() {
           return mark;
    }

    public double getSumTime() {
           return sumTime;
    }

    public void incCountAppear() {
           countAppear++;
    }

    public void incSumTime(double time) {
           sumTime = sumTime + time;
    }

    /**
     * @param mark
     *            the mark to set
     */
    public void setMark(Marking mark) {
           this.mark = mark;
    }

@Override
public boolean equals(Object obj) {
    StatLine sta = (StatLine) obj;
    return sta.getMark().equals(this.getMark()) ;
}

public void incLackTime(double sumlack) {
    this.lackTime += sumlack;
}
```

```
    public double getLackTime() {
        return lackTime;
    }

    public void setLastTime(double lastTime) {
        this.lastTime = lastTime;
    }

    public double getLastTime() {
        return lastTime;
    }

    public double getAvgSummTime() {
        return getSumTime()/getCountAppear();
    }

    public double getAvgBacktime() {
        return getLackTime()/getCountAppear();
    }
}
```

# \processing

## \processing\BuildTree.java

```
package
 ua.com.handcrafted.petri.processing;

import java.util.ArrayList;

import ua.com.handcrafted.petri.data.Bridge;
import ua.com.handcrafted.petri.data.DataModel;
import ua.com.handcrafted.petri.matrix.Vector;

/**
 * @author Pustovit Michael
 * Построение дерева достижимости.
 */

public class BuildTree {
       //Начальная маркировка
       public Marking startMarking;

       DataModel model;


       /**
        * Конструктор построителя дерева достижимости.
        * @param startMarking Начальная маркировка.
        * @param model Модель данных сети, дерево которой строится.
        */
       public BuildTree (Marking startMarking, DataModel model){
              this.startMarking = startMarking;
              this.startMarking.setStatus(MarkingStatus.ROOT);
              this.model = model;
       }

       /**
        * Получение следующей маркировки.
        * @param currentMark Текущая маркировка.
        * @param vectBridg Вектор разрешенных переходов.
        * @return Новая маркировка.
        */
       private Marking getNextMarking(Marking currentMark, Vector vectBridg) {
           Vector vec2 = vectBridg.multiple(model.getResultMatrix());
           int [] vec1 = currentMark.getMarkingVector().add(vec2).getVector();
           return new Marking(vec1);
```

```
	}

	/**
	 * Построение сына маркироки currentMark после совершения переходов currentVect.
	 * @param prevMark Текущая маркировка.
	 * @param currentVect Вектор разрешенных переходов.
	 * @param wasthere Вектор ранее полученных маркировок.
	 * @return Маркировка после совершения всех разрешенных переходов.
	 */
	private ArrayList<Marking> getSon(Marking prevMark, Vector currentVect,
ArrayList<Vector> wasthere) {
		ArrayList<Vector> curwasthere = new ArrayList<Vector>();
		Marking tprevMark = new Marking(prevMark.getMarking());
		ArrayList<Marking> currentMarks = getInstantSon(tprevMark, currentVect,
curwasthere);

		for (int z = 0; z < currentMarks.size(); z++) {
		if (!wasthere.contains(currentMarks.get(z).getMarkingVector())) {
		    wasthere.add(currentMarks.get(z).getMarkingVector());

		    model.setMarking(currentMarks.get(z));
		    ArrayList<Bridge> bridges = model.getAllowTimeBridges();

		    //prevMark.addChildren(currentMarks);
		    Marking cM;
		       if (bridges.size() > 0) {
		       for (int i = 0; i < bridges.size(); i++) {
		           currentVect.clear();
		           currentVect.getVector()[model.getBridgeIndex(bridges.get(i))]++;

		           cM = currentMarks.get(z);
		           cM.addChildren(getSon(cM, currentVect, wasthere));
		       }
		       } else {
		           currentMarks.get(z).setStatus(MarkingStatus.DEADLOCK);
		       }
		} else {
		    currentMarks.get(z).setStatus(MarkingStatus.DUBLICATE);
		}
		}
		return currentMarks;
	}

	/**
	 * @param prevMark Маркировка из которой совершается  переход.
	 * @param currentVect Вектор переходов которые разрешены.
	 * @param curwasthere Массив маркировок в которых мы уже побывали.
	 * @return Массив последующих маркировок (их может быть несколько из за
конфликтов
	 * мгновенных переходов, хотя по условию решаемой задачи такой случай исключен).
	 * Получение следующей значащей маркировки из текущей маркировки и вектора
разрешенных
	 * переходов.
	 */
	public ArrayList<Marking> getInstantSon(Marking prevMark, Vector currentVect,
		ArrayList<Vector> curwasthere) {
	    ArrayList<Marking> currentMarks = new ArrayList<Marking> ();


	    if (!currentVect.isEmpty()) {
		Marking tempMark;
		Vector tempVect;
		for (int i = 0; i < currentVect.getDimension(); i++) {
		   if (currentVect.getVector()[i] == 1) {
		       tempMark = getNextMarking(prevMark, new Vector(i,
currentVect.getDimension()));
		       ArrayList<Bridge> temp = prevMark.getPath();
		       temp.add(model.getBridgeByIndex(i));
		       tempMark.setPath(temp); //prevMark.getStrPath() +
model.getBridgeByIndex(i).getId() + " "
```

```
                    int border;
                    for (int j = 0; j < tempMark.getMarking().size(); j++) {
                         border = model.getStateByIndex(j).getMaxChips();
                         if (border != -1) {
                             if (tempMark.getMarking().get(j) > border) {
                                 tempMark.getMarking().set(j, border);
                             }
                         }
                      }

                    model.setMarking(tempMark);
                           if (!curwasthere.contains(tempMark.getMarkingVector())) {
                               curwasthere.add(tempMark.getMarkingVector());
                               tempVect = model.getAllowInstantBridgesVector();
                           currentMarks.addAll(getInstantSon(tempMark, tempVect,
curwasthere));
                           } else {
                               currentMarks.add(tempMark);
                           }
                }
            }

             for (int i = 0; i < currentMarks.size(); i++) {
                 for (int j = i + 1; j < currentMarks.size(); j++) {
                     if (currentMarks.get(i).equals(currentMarks.get(j))) {
                         currentMarks.remove(j);
                     }
                 }
             }
          } else {
              currentMarks.add(prevMark);
          }
          return currentMarks;
        }

        /**
         * Print into console tree in text format.
         * @param tree Tree of accessibility tree.
         * @param ots Indent of previos node (top node have indent "",
         *     node of second level - "  " etc.)
         */
        public void treePrint(Marking tree, String ots) {
            System.out.println(ots + tree + "  "+ tree.getStatus());
            for (int i = 0; i < tree.getChildren().size(); i++) {
                treePrint(tree.getChildren().get(i), ots + "  ");
            }
        }

        /**
         * Поиск маркировки в дереве начинающемся с firstMark.
         * @param firstMark Начальная маркировка дерева.
         * @param curMark Искомая маркировка.
         * @return Искомый узел дерева.
         */
        public Marking findMarking(Vector firstMark, Marking curMark) {
            if (firstMark.equals(curMark.getMarkingVector()) &&
                     (curMark.getStatus() == MarkingStatus.INNER)) {
                return curMark;
            } else {
                Marking tempMark;
                for (int i = 0; i < curMark.getChildren().size(); i++) {
                    tempMark = findMarking(firstMark, curMark.getChildren().get(i));
                    if (tempMark != null) {
                        return tempMark;
                    }
                }
            }
            return null;
        }
```

```
        /**
         * Преобразование дерева достижимости к ярусно-паралельной форме.
         * @param tree Дерево достижимости.
         * @param wasthere Массив вершин где мы уже побывали.
         * @return Узел дерева из которого начинается дерево в ярусно-паралельной форме.
         */
        public Marking treeLifter(Marking tree, ArrayList<Vector> wasthere) {
            wasthere.add(tree.getMarkingVector());
            for (int i = 0; i < tree.getChildren().size(); i++) {
                if ((tree.getChildren().get(i).getStatus() == MarkingStatus.DUBLICATE)
&&

!wasthere.contains(tree.getChildren().get(i).getMarkingVector())) {
                    Marking findMark =
findMarking(tree.getChildren().get(i).getMarkingVector(), tree);
                    if (findMark != null) {
                    ArrayList<Bridge> path = tree.getChildren().get(i).getPath();
                    tree.getChildren().set(i,findMark.clone());
                    tree.getChildren().get(i).setPath(path);
                    findMark.clearChildren();
                    findMark.setStatus(MarkingStatus.DUBLICATE);
                    treeLifter(tree.getChildren().get(i), wasthere);
                    }
                }
            }
            return tree;
        }

        /**
         * Start building of accessibility tree.
         * @return Top node of accessibility tree.
         */
        public ArrayList<Marking> startTreeBuild() {
            ArrayList<Marking> tree;
            if (!model.isStateListEmpty()) {
            model.setMarking(startMarking);

            Vector vect = model.getAllowInstantBridgesVector();

            ArrayList<Vector> allVect = new ArrayList<Vector>();

            tree = getSon(startMarking, vect, allVect);

            for (int i = 0; i < tree.size(); i++) {
                tree.get(i).setStatus(MarkingStatus.ROOT);
            }

            tree.set(0, treeLifter(tree.get(0), new ArrayList<Vector>()));

            /*for (int i = 0; i < tree.size(); i++) {
                treePrint(tree.get(i), "");
            }*/
            } else {
                System.out.println("В модели нет состояний. Построение дерева
невозможно.");
                tree = new ArrayList<Marking>();
                tree.add(new Marking(new int [] {0}));
            }
            return tree;
        }
}
```

## \processing\Marking.java

```
package ua.com.handcrafted.petri.processing;

import java.util.ArrayList;
```

```
import ua.com.handcrafted.petri.data.Bridge;
import ua.com.handcrafted.petri.matrix.Vector;

/**
 * @author Hancrafted. Kirill Novotarsky
 * Класс маркировки. Содержит в себе общую маркировку сети Петри
 * в некоторый момент, а также список активных временных переходов
 * (которые могут быть выполнены при текущей маркировке).
 * Нужно учесть, что маркировка является композитной структурой -
 * она содержит в себе ссылку на родительскую маркировку и на детей
 * Это сделано для того чтоб потом разобрать построенный лист с маркировками
 */
public class Marking {
    /**
     * Маркировка
     */
    ArrayList<Integer> marking;

    /**
     * Активные переходы
     */
    ArrayList<Bridge> transitions = new ArrayList<Bridge>();

    /**
     * Имя маркировки
     */
    private String name = new String();

    /**
     * Status of marking. Can be:
     * "Тупиковая"
     * "Повторяющаяся"
     * "Корневая"
     * "Внутренняя"

     */
    private String status = new String();

    /**
     *  Ссылка на родителя маркировки
     */
    Marking parent;

    /**
     * Список детей маркировки
     */
    ArrayList<Marking> children = new ArrayList<Marking>();

    /**
     * Путь по которому мы попали в вершину (совершенные переходы).
     */
    private ArrayList<Bridge> path = new ArrayList<Bridge>();

    /**
     * Конструктор принимающий ArrayList элементов типа Integer.
     * @param mark Маркировка которой будет инициализированна маркировка.
     */
    public Marking(ArrayList<Integer> mark) {
        marking = new ArrayList<Integer>();
        setMarking(mark);
        this.name = toString();
        this.setStatus(MarkingStatus.INNER);
    }

    /**
     * Конструктор принимающий статический массив целых значений.
     * @param mark Маркировка которой будет инициализированна маркировка.
     */
    public Marking(int [] mark) {
        marking = new ArrayList<Integer>();
```

```
    setMarking(mark);
    this.name = toString();
    this.setStatus(MarkingStatus.INNER);
}

/**
 * Получаем путь до маркировки в стринге
 * @return
 */
public String getStrPath() {
    String strpath = "";
    for (int i = 0; i < path.size(); i++) {
        strpath = strpath + path.get(i).getId() + " ";
    }
    return strpath;
}

/**
 * Путь списком бриджей
 * @return
 */
public ArrayList<Bridge> getPath() {
    return path;
}

/**
 * Сеттер для пути
 * @param path
 */
public void setPath(ArrayList<Bridge> path) {
    this.path = path;
}

/**
 * Возвращает значение поля "маркировка".
 * @return Вектор данной маркировки.
 */
public ArrayList<Integer> getMarking(){
    return this.marking;
}

/**
 * Геттер для вектора маркировки
 * @return вектор
 */
public Vector getMarkingVector() {
    int [] rez = new int [marking.size()];
    for (int i = 0; i < marking.size(); i++) {
        rez[i] = marking.get(i);
    }
    return new Vector(rez);
}

/**
 * Сеттер для маркировки листом целых
 * @param marking
 */
public void setMarking(ArrayList<Integer> marking){
        this.marking = marking;
}

/**
 * Сеттер для маркировки массивом целых
 * @param marking
 */
public void setMarking(int [] marking){
    this.marking.clear();
    for (int i = 0; i < marking.length; i++) {
        this.marking.add(marking[i]);
    }
```

```
}

/**
 * Геттер бриджей
 * @return
 */
public ArrayList<Bridge> getTransitions(){
    return this.transitions;
}

/**
 * Сеттер для бриджей
 * @param transitions
 */
public void setTransitions(ArrayList<Bridge> transitions){
    this.transitions = transitions;
}

/**
 * Добавить маркировке ребенка
 * @param mark - маркировка-ребенок
 */
public void addChild(Marking mark) {
    children.add(mark);
}

/**
 * Добавить маркировке список детей
 * @param mark -список маркировок
 */
public void addChildren(ArrayList<Marking> mark) {
    children.addAll(mark);
}

/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
public String toString() {
    String str = "";
    for (int i = 0; i < marking.size(); i++) {
        str = str + marking.get(i)+ " ";
    }
    return str;
}

/**
 * Геттер для детей маркировки
 * @return
 */
public ArrayList<Marking> getChildren(){
    return this.children;
}

/**
 * Очистить список детей маркировки
 */
public void clearChildren() {
    children.clear();
}

/**
 * Вернуть имя маркирвоки
 * @return Имя
 */
public String getName() {
    return name;
}

/**
 * Сеттер для имени
```

```
     * @param str - имя
     */
    public void setName(String str) {
        name = str;
    }

    /**
     * Геттер для статуса маркировки
     * @return
     */
    public String getStatus() {
        return status;
    }

    /**
     * Сеттер для статуса маркировки
     * @param status
     */
    public void setStatus(String status) {
        this.status = status;
    }

    /* (non-Javadoc)
     * @see java.lang.Object#clone()
     */
    public Marking clone() {
        Marking tempMark = new Marking(marking);
        tempMark.addChildren(children);
        tempMark.setStatus(status);
        tempMark.setName(name);
        return tempMark;
    }

    /* (non-Javadoc)
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(Object obj) {
        Marking temp = (Marking) obj;
        return temp.getMarking().equals(this.marking);
    }

}
```

## \processing\MarkingStatus.java

```
package ua.com.handcrafted.petri.processing;

/**
 * @author Pustovit Michael
 * Все возможные состояния маркировки в дереве достижимости.
 */
public class MarkingStatus {
    public static String ROOT = "Корневая";
    public static String INNER = "Внутреняя";
    public static String DUBLICATE = "Дублирующая";
    public static String DEADLOCK = "Тупиковая";

}

\tree
\tree\TreeVis.java
package ua.com.handcrafted.petri.tree;
import java.util.ArrayList;
import javax.swing.JPanel;
import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
```

```
import javax.swing.UIManager;

import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;

import sun.applet.Main;
import ua.com.handcrafted.petri.data.Bridge;
import ua.com.handcrafted.petri.data.DataModel;
import ua.com.handcrafted.petri.data.State;
import ua.com.handcrafted.petri.data.TimeBridge;
import ua.com.handcrafted.petri.processing.BuildTree;
import ua.com.handcrafted.petri.processing.Marking;

import java.net.URL;
import java.io.IOException;
import java.awt.Dimension;
import java.awt.GridLayout;

/**
 * Класс дерева достижимости слева от графического представления
 * @author Кирилл Новотарский
 *
 */
public class TreeVis extends JPanel
implements TreeSelectionListener {

       //private JEditorPane textPane;
    /**
     *Класс собственно отображаемого дерева
     */
    private JTree tree;
    /**
     *  Маркировка (со всеми наследниками), которую мі отображаем
     */
    private Marking whatAreWeShowing;

    /**
     *Название дерева
     */
    DefaultMutableTreeNode top =
        new DefaultMutableTreeNode("Our Tree");
    private static boolean DEBUG = false;

    //Optionally play with line styles.  Possible values are
    //"Angled" (the default), "Horizontal", and "None".
    private static boolean playWithLineStyle = false;
    private static String lineStyle = "Horizontal";

    //Optionally set the look and feel.
    private static boolean useSystemLookAndFeel = false;

    /**
     * Обновление всего дерева
     * @param main - маркировка нового дерева
     */
    public void doNewTree (Marking main){
       top.removeAllChildren();
       DefaultMutableTreeNode global = new DefaultMutableTreeNode(main.getName());
       top.add(global);
       createNodes(global, main);
    }

    /**
     * Тестовый метод для получения тестовой модели данный по Сети, необязателен
     * @return тестовая модель
     */
```

```
    public DataModel getTestDataModel() {
       //TODO Delete in future
            DataModel model = new DataModel();

            State st1 = new State();
            State st2 = new State();
            State st3 = new State();
            State st4 = new State();
            State st5 = new State();
            model.addState(st1);
            model.addState(st2);
            model.addState(st3);
            model.addState(st4);
            model.addState(st5);

            Bridge br1 = new Bridge();
            TimeBridge br2 = new TimeBridge();
            TimeBridge br3 = new TimeBridge();
            TimeBridge br4 = new TimeBridge();
            TimeBridge br5 = new TimeBridge();

            br1.addInLink(st1);
            br1.addOutLink(st2);
            br1.addInLink(st4);
            br2.addInLink(st2);
            br2.addOutLink(st3);
            br3.addInLink(st3);
            br3.addOutLink(st4);
            br4.addInLink(st3);
            br4.addOutLink(st5);
            br5.addInLink(st5);
            br5.addOutLink(st2);

            model.addBridge(br1);
            model.addBridge(br2);
            model.addBridge(br3);
            model.addBridge(br4);
            model.addBridge(br5);

            model.initializeMatrix();
            return model;
       }

    /**
     * Получаем тестовую маркировку
     * @return маркировка
     */
    public Marking getTestMarking() {
       //TODO Delete in future
       int [] mar = {1, 0, 1, 0, 0};
         Marking mark  = new Marking(mar);
       return mark;
    }


       /**
        * Конструктор нашего дерева - получаем модель данных и маркировку - и все
строим
        * @param model - передаваемая модель
        * @param main - передаваемая маркировка
        */
       public TreeVis(DataModel model, Marking main) {
        super(new GridLayout(1,0));

        if (model == null){
             model = getTestDataModel();
        }

        Marking mark;
```

```
        if (main == null){

             mark = this.getTestMarking();
        } else {
             mark = main;
        }

        BuildTree proc = new BuildTree(mark, model);

        ArrayList<Marking> newMark
 = new ArrayList<Marking>();
        newMark = proc.startTreeBuild();
        for (int i = 0; i < newMark.size(); i++){
        DefaultMutableTreeNode global = new
DefaultMutableTreeNode(newMark.get(i).getName());
        top.add(global);
        createNodes(global, newMark.get(i));
        }
        //Create a tree that allows one selection at a time.
        tree = new JTree(top);
        tree.getSelectionModel().setSelectionMode
               (TreeSelectionModel.SINGLE_TREE_SELECTION);

        //Listen for when the selection changes.
        tree.addTreeSelectionListener(this);

        if (playWithLineStyle) {
            System.out.println("line style = " + lineStyle);
            tree.putClientProperty("JTree.lineStyle", lineStyle);
        }

        JScrollPane treeView = new JScrollPane(tree);

       Dimension minimumSize = new Dimension(100, 50);
       treeView.setMinimumSize(minimumSize);

        add(treeView);
    }


    /** Required by TreeSelectionListener interface. */
    public void valueChanged(TreeSelectionEvent e) {
        DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                           tree.getLastSelectedPathComponent();

        if (node == null) return;

        Object nodeInfo = node.getUserObject();
        if (node.isLeaf()) {

            int a=0;
            }  else {
        }
        if (DEBUG) {
            System.out.println(nodeInfo.toString());
        }
    }


    /**
     * Преобразовует структуру маркировки во внутреннюю структуру деревя
     * @param top
     * @param struct
     */
    private void createNodes(DefaultMutableTreeNode top, Marking struct) {
        DefaultMutableTreeNode category = null;
        for (int i = 0; i< struct.getChildren().size(); i++){

              category = new
DefaultMutableTreeNode(struct.getChildren().get(i).getName());
```

```
            top.add(category);

            createNodes(category, struct.getChildren().get(i));
       }
         }
       //  top.removeAllChildren();



    /**
     * Сеттер для маркировок
     * @param whatAreWeShowing
     */
    public void setWhatAreWeShowing(Marking whatAreWeShowing) {
            this.whatAreWeShowing = whatAreWeShowing;
       }

       /**
        * Геттер для маркировки
        * @return
        */
       public Marking getWhatAreWeShowing() {
            return whatAreWeShowing;
       }

       /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event dispatch thread.
     */
    private static void createAndShowGUI() {
        if (useSystemLookAndFeel) {
            try {
                UIManager.setLookAndFeel(
                    UIManager.getSystemLookAndFeelClassName());
            } catch (Exception e) {
                System.err.println("Couldn't use system look and feel.");
            }
        }

        //Create and set up the window.
        JFrame frame = new JFrame("Дерево достижимости ");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Add content to the window.

       frame.add(new TreeVis(null,null));

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    /**
     * Метол для тестирования дерева
     * @param args
     */
    public static void main(String[] args) {
        //Schedule a job for the event dispatch thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```